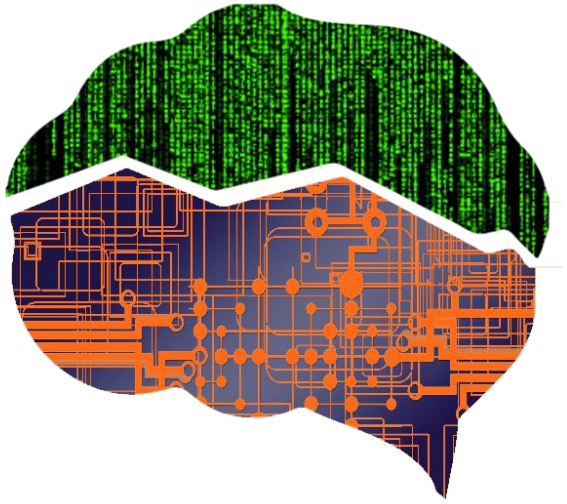
Software Product Line Engineering

Runtime Variability

Christian Kästner (Carnegie Mellon University)
Sven Apel (Universität Passau)
Norbert Siegmund (Bauhaus-Universität Weimar)
Gunter Saake (Universität Magdeburg)



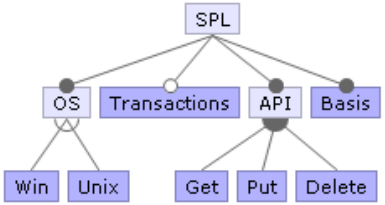
Bauhaus-Universität Weimar

How to implement variability?

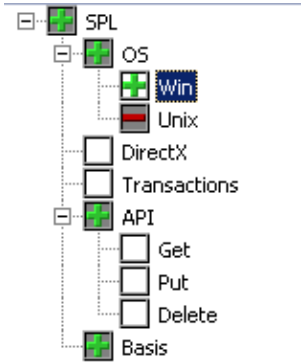
Domain Eng.

Application Eng.

Feature Model

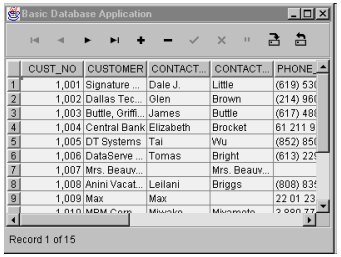


Reusable Implementation artifacts



Feature Selection

Generator



Resulting Program

Agenda

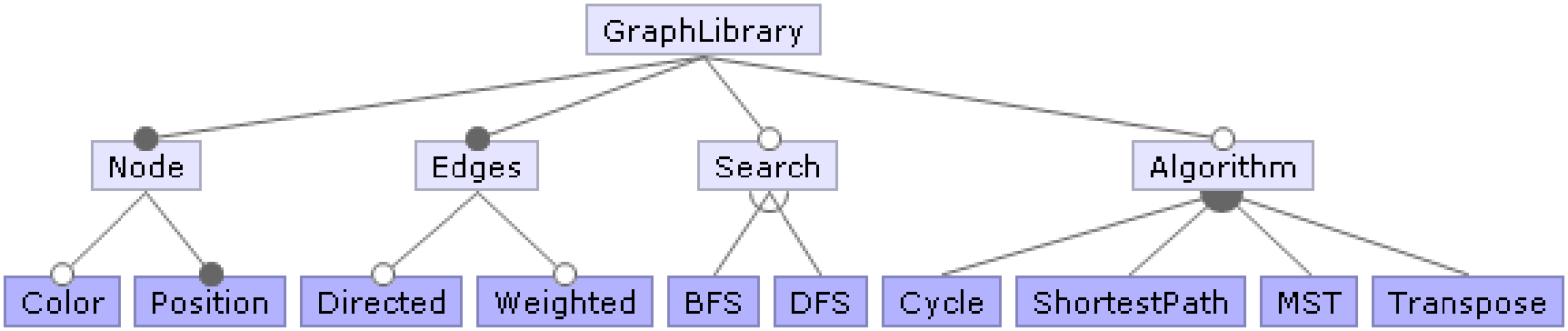
- Graph example
- Variability with runtime parameter
- Recap: Modularity
- Design patterns for variability
- Limitations of traditional techniques

An Example

Example: Graph Library

- Running example in lecture
  (Chat-Client in exercise)
- Library of graph data structures and algorithms
  - Weighted/unweighted edges
  - Directed/undirected edges
  - Colored nodes
  - Algorithms: shortest path, minimal spanning tree, transitive closure, …

Graph Feature Model



Example: Graph Implementation

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Runtime Parameter

Parameter

~

Main Options VT Options VT Fonts

```
apel@Zerberus ~
$ grep --help
Aufruf: grep [OPTION]... MUSTER [DATEI]...
Suche nach MUSTER in jeder DATEI oder der Standardeingabe.
MUSTER ist normalerweise ein regulärer Standardausdruck (BRE).
Beispiel: grep -i 'Hallo Welt' menu.h main.c

Auswahl und Interpretation regulärer Ausdrücke:
  -E, --extended-regexp     PATTERN is an extended regular expression (ERE)
  -F, --fixed-strings       PATTERN is a set of newline-separated strings
  -G, --basic-regexp        PATTERN is a basic regular expression (BRE)
  -P, --perl-regexp         PATTERN is a Perl regular expression
  -e, --regexp=MUSTER       MUSTER als regulären Ausdruck verwenden.
  -f, --file=FILE           MUSTER aus DATEI lesen.
  -i, --ignore-case         Unterschied zwischen Groß- und Kleinschreibung
                            ignorieren.
  -w, --word-regexp         MUSTER passt nur auf ganze Wörter.
  -x, --line-regexp         MUSTER passt nur auf ganze Zeilen.
  -z, --null-data           Eine Zeile endet mit Nullbyte, nicht Newline.

Verschiedenes:
  -s, --no-messages         Fehlermeldungen unterdrücken.
  -v, --invert-match        Nicht-passende Zeilen anzeigen.
  -V, --version             Versionsnummer ausgeben und beenden.
      --help                Diese Hilfe ausgeben und beenden.

Kontrolle der Ausgabe:
  -m, --max-count=ANZAHL    stoppt nach ANZAHL Übereinstimmungen
  -b, --byte-offset         gibt mit den Zeilen auch den Abstand in Bytes an
  -n, --line-number         gibt mit den Zeilen auch die Zeilennummer an
      --line-buffered       leert den Puffer nach jeder Zeile
  -H, --with-filename       gibt den Dateinamen für jede Übereinstimmung aus
  -h, --no-filename         unterdrückt die Ausgabe des vorangehenden Dateinamens
      --label=BEZEICHNUNG   verwendet BEZEICHNUNG als Präfix für Dateinamen
                            der Standardeingabe
  -o, --only-matching       zeigt nur den Teil einer Zeile, der zu MUSTER passt
  -q, --quiet, --silent     unterdrückt alle normalen Ausgaben
      --binary-files=TYP    alle binären Dateien sind vom Typ TYP;
                            TYP kann  binary",  text" oder  without-match" sein
  -a, --text                gleichbedeutend mit --binary-files=text
  -I                        gleichbedeutend mit --binary-files=without-match
  -d, --directories=AKTION  beschreibt, wie Verzeichnisse zu behandeln sind;
                            AKTION kann  read",  recurse" oder  skip" sein
  -D, --devices=AKTION      Behandlung von Geräten, FIFOs oder Sockets;
                            AKTION kann  read" oder  skip" sein
  -r, --recursive           wie --directories=recurse
  -R, --dereference-recursive  ebenso, folgt aber allen symbolischen Links
      --include=DATEIMUSTER  durchsucht nur Dateien, die DATEIMUSTER entsprechen
      --exclude=DATEIMUSTER  überspringt Dateien und Verzeichnisse, die
                              DATEIMUSTER entsprechen
      --exclude-from=DATEI   überspringt Dateien, die einem Dateimuster
                              in DATEI entsprechen
      --exclude-dir=MUSTER   Verzeichnisse, die MUSTER entsprechen, werden übersprungen
  -L, --files-without-match  nur die Namen von Dateien ausgeben, die keinen
                              passenden Inhalt haben
  -l, --files-with-matches  nur die Namen von Dateien mit passendem Inhalt
                            ausgeben
  -c, --count               nur die Anzahl der passenden Zeilen pro DATEI
                            ausgeben
```

Parameter –i in grep

```
1  int match_icase;
2
3  int main (int argc, char **argv)
4  {
5    [...]
6    while ((opt = get_nondigit_option (argc, argv, &default_c
7      switch (opt)
8        {
9        [...]
10       case 'i':
11         match_icase = 1;
12         break;
13       }
14   }
15
16
17 static const char *
18 print_line_middle (const char *beg, const char *lim,
19                    const char *line_color, const char *match_color)
20 {
21   [...]
22   if (match_icase)
23     {
24       ibeg = buf = (char *) xmalloc(i);
25       while (--i >= 0)
26         buf[i] = tolower(beg[i]);
```

Global Configuration Options

```
class Conf {
   public static boolean Logging = false;
   public static boolean Windows = false;
   public static boolean Linux = true;
}
class Main {
  public void foo() {
     if (Conf.Logging)
       log(„running foo()“);
     if (Conf.Windows)
       callWindowsMethod();
     else if (Conf.Linux)
       callLinuxMethod();
     else
       throw RuntimeException();
  }
}
```

Graph Implementation

```
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}
```

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

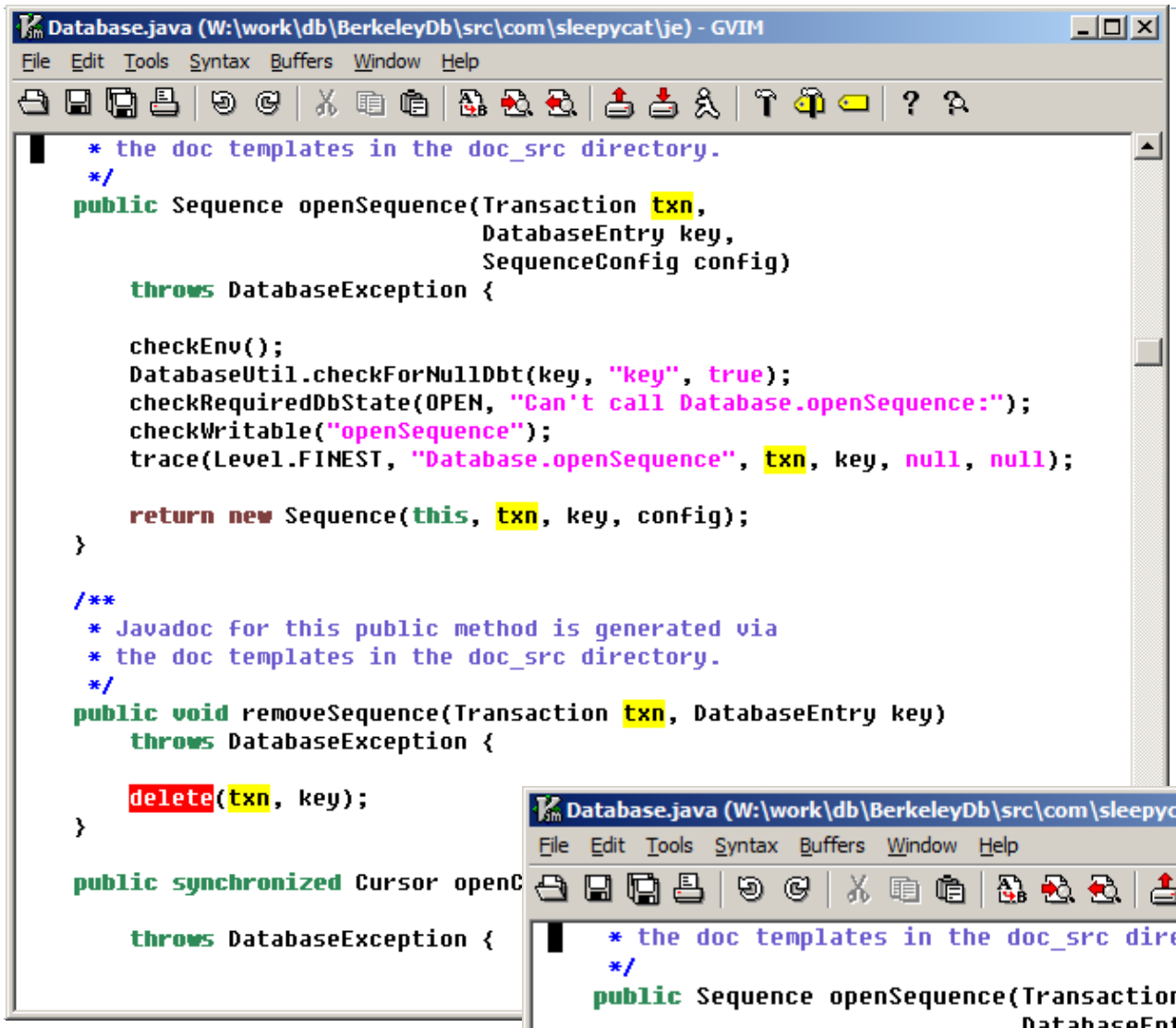
```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Propagating Parameter



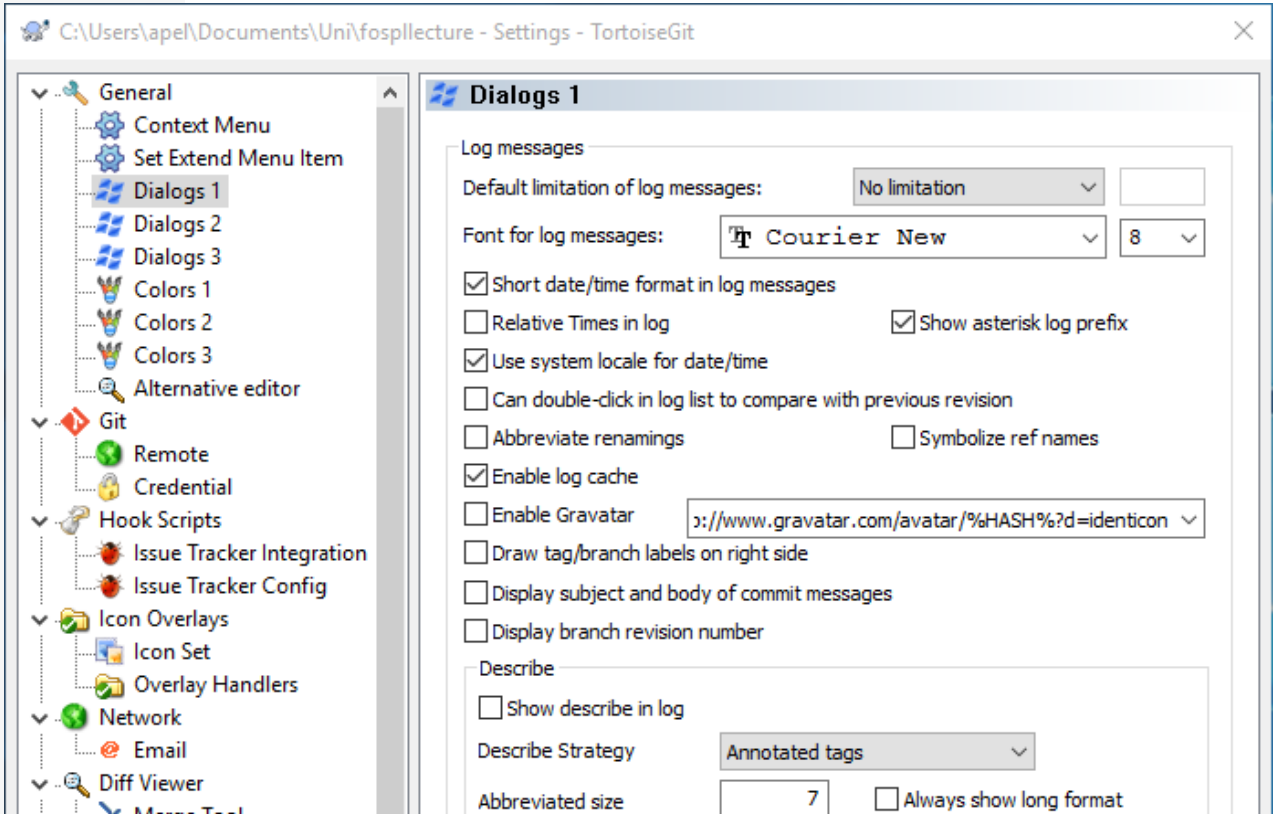
Propagating through many calls instead of global variable

Configuration

- Command-line parameter
- Config file
- Dialog
- Source code
- Registry
- …

httpd.conf -- win32 Apache
Building a Web Server, for Windows

```
Listen 80
ServerRoot "/www/Apache2"
DocumentRoot "/www/webroot"

ServerName localhost:80
ServerAdmin admin@localhost

ServerSignature On
ServerTokens Full

DefaultType text/plain
AddDefaultCharset ISO-8859-1

UseCanonicalName Off

HostnameLookups Off

ErrorLog logs/error.log
LogLevel error

PidFile logs/httpd.pid

Timeout 300
```



Discussion

- Variability distributed over the whole program code
- Either global variables or long parameter lists

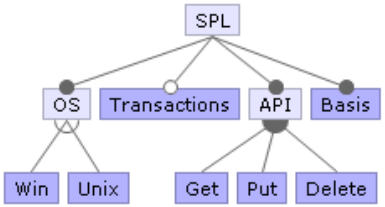
- Configuration verified?
- Is reconfiguration possible at runtime?
- Protected from calls of deactivated functionality?

- No generator; always the whole variability is deployed
  - Code size, resource consumption, performance
  - Unused functionality as security risk

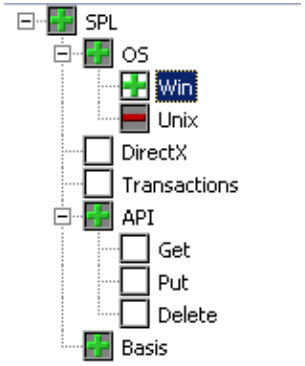
Domain Eng.

Application Eng.

Parameter list
(feature model)

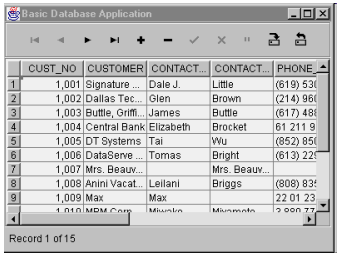


Program with
runtime parameters



Parameter selection
(feature selection)

Setting the start parameters



Program execution
with desired behavior

Recap: Modularity

What is Modularity?

- Modularity is cohesion with encapsulation
- **Encapsulation**: Hiding of implementation details behind an interface
- **Cohesion**: Grouping related program constructs in one unit (e.g., package, class, …)
- Cohesive and weakly coupled modules can be understood in an isolated way
- Reduces complexity of the development process

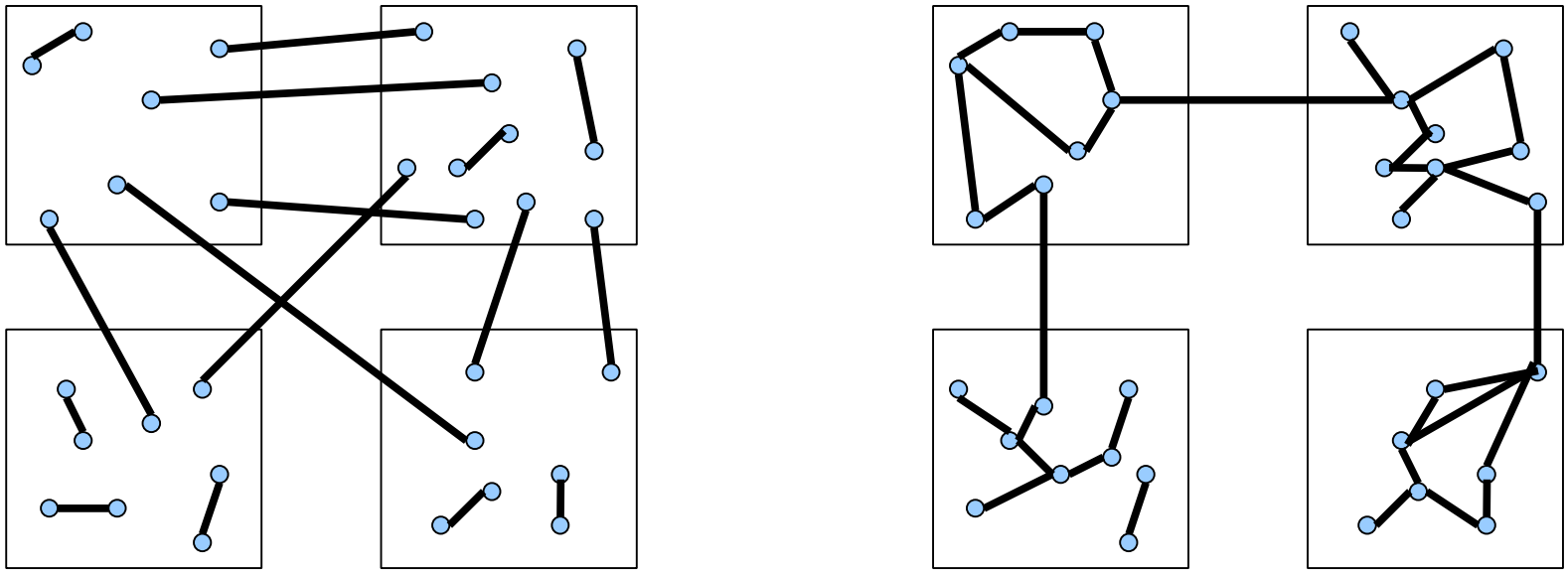
Encapsulation

```
public class ArrayList<E> {
   public void add(int index, E element) {
      if (index > size || index < 0)
         throw new IndexOutOfBoundsException(
            "Index: "+index+", Size: "+size);
      ensureCapacity(size+1);
      System.arraycopy(elementData, index,
         elementData, index + 1, size - index);
      elementData[index] = element;
      size++;
   }
   public int indexOf(Object o) {
      if (o == null) {
         for (int i = 0; i < size; i++)
            if (elementData[i]==null)
            return i;
      } else {
         for (int i = 0; i < size; i++)
            if (o.equals(elementData[i]))
            return i;
      }
      return -1;
    }
   ……..
}
```

```
public interface List<E> {
   void add(int index, E element);
   int indexOf(Object o);
   ….
}
```

- Hide implementation details
- Interface describes behavior
- Implementation can be replaced

Cohesion/Coupling -- Example

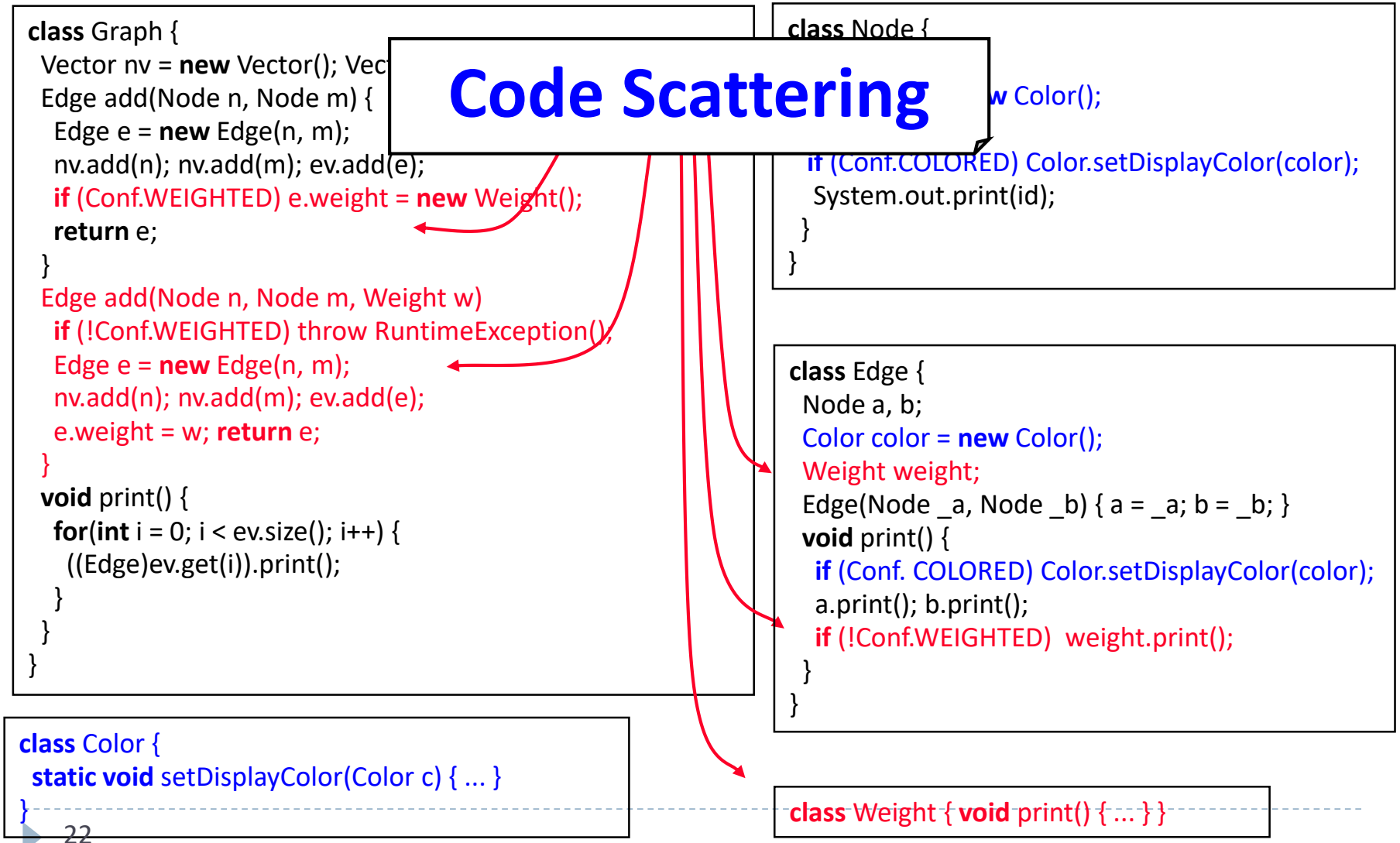


- Grouping of methods and tasks
- Many calls over group boundaries
- Group implements different features

Why Modularity?

- Software easy to understand(*divide and conquer*)
- Hides complexity from parts of the software behind interface (*information hiding*)
- Easier to maintain, since all changes can be made locally (*maintainability*)
- Parts of the software can be reused (*reusability*)
- Modules can be composed within a new context of another project (*variability*)

Problems? – Scattered Code



Problems? – Tangled Code

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    ...
  }
}
```

Code Tangling

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Problems? – Replicated Code

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, int w) {
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

Code Replication

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

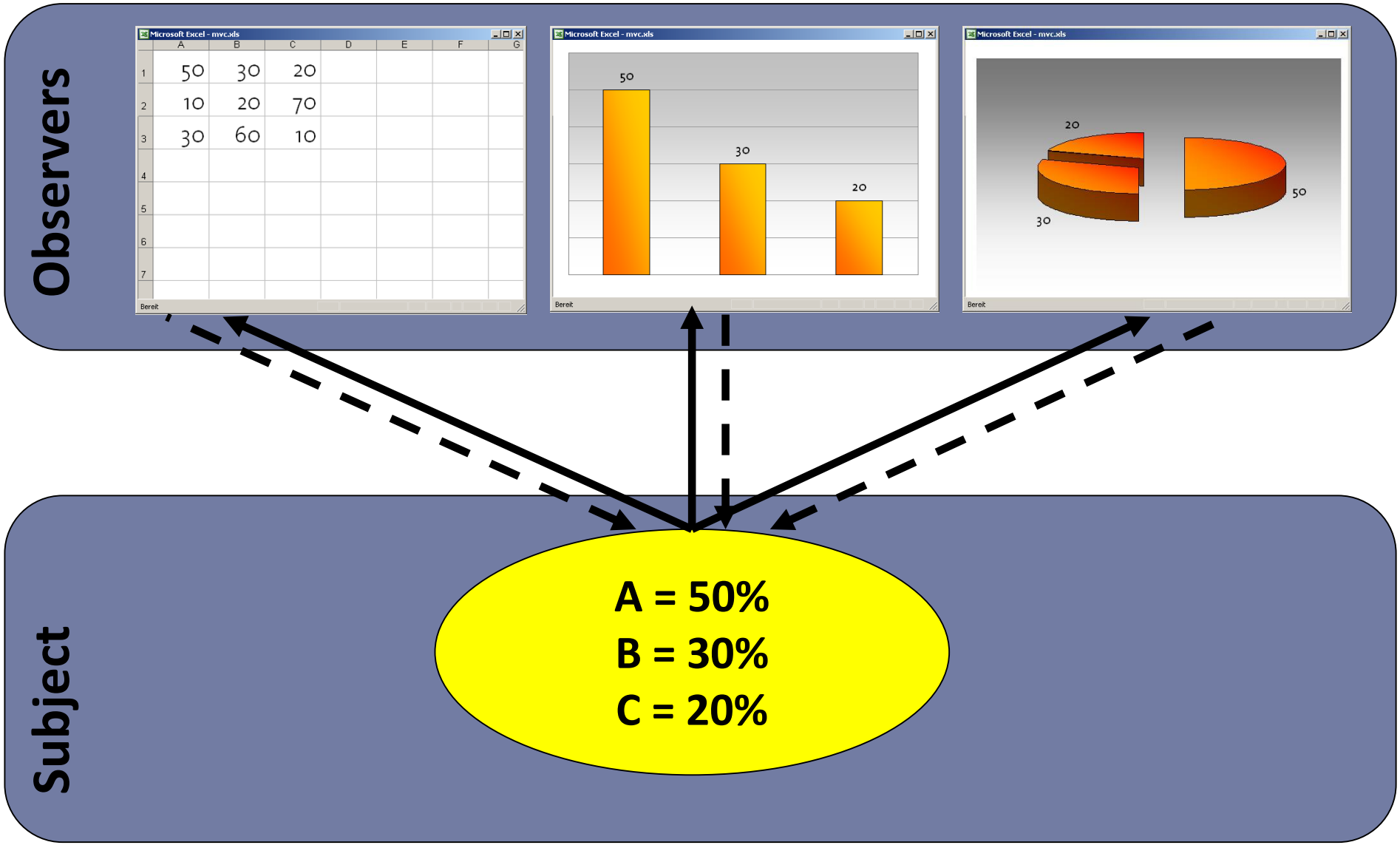
Design Patterns for Variability

Design Patterns

- Pattern for the design of a solution of a reoccurring problem
- Many design patterns to achieve variability, extendibility, and decoupling of functionality

- Selection:
  - Observer
  - Template Method
  - Strategy
  - Decorator

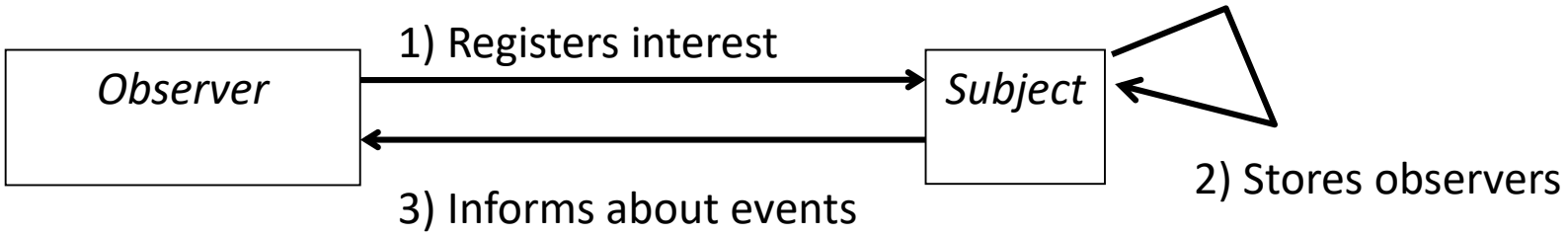
Observer Pattern



[Quelle: Meyer/Bay]

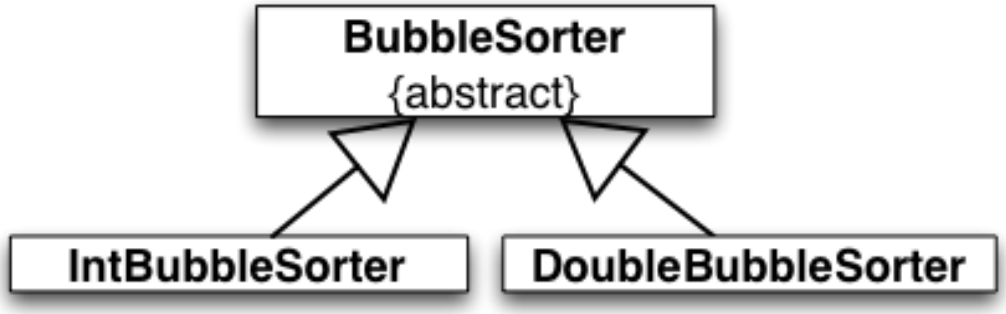
Observer Pattern

"*Define*[*s*] *a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.*" [GoF, p 293]



- Interfaces add extra flexibility
- In implementation
  - Observer class(es)
  - Subject class
  - List in Subject storing registered Observer
  - Subject.AddToObservers(Observer) (called by Observer)
  - Observer.notify() (called by Subject)

Template Method Pattern

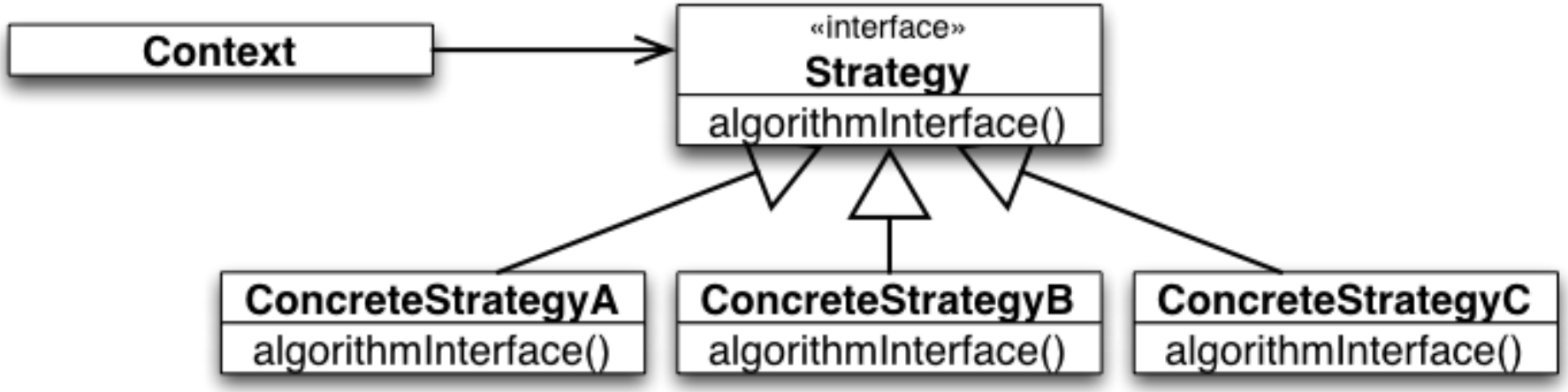



```
public abstract class BubbleSorter{
       protected int length = 0;
       protected void sort() {
              if (length <= 1) return;
              for (int nextToLast= length-2;
                     nextToLast>= 0; nextToLast--)
                     for (int index = 0;
                            index <= nextToLast; index++)
                            if (outOfOrder(index)) swap(index);
       }
       protected abstract void swap(int index);
       protected abstract boolean outOfOrder(int index);
}
```

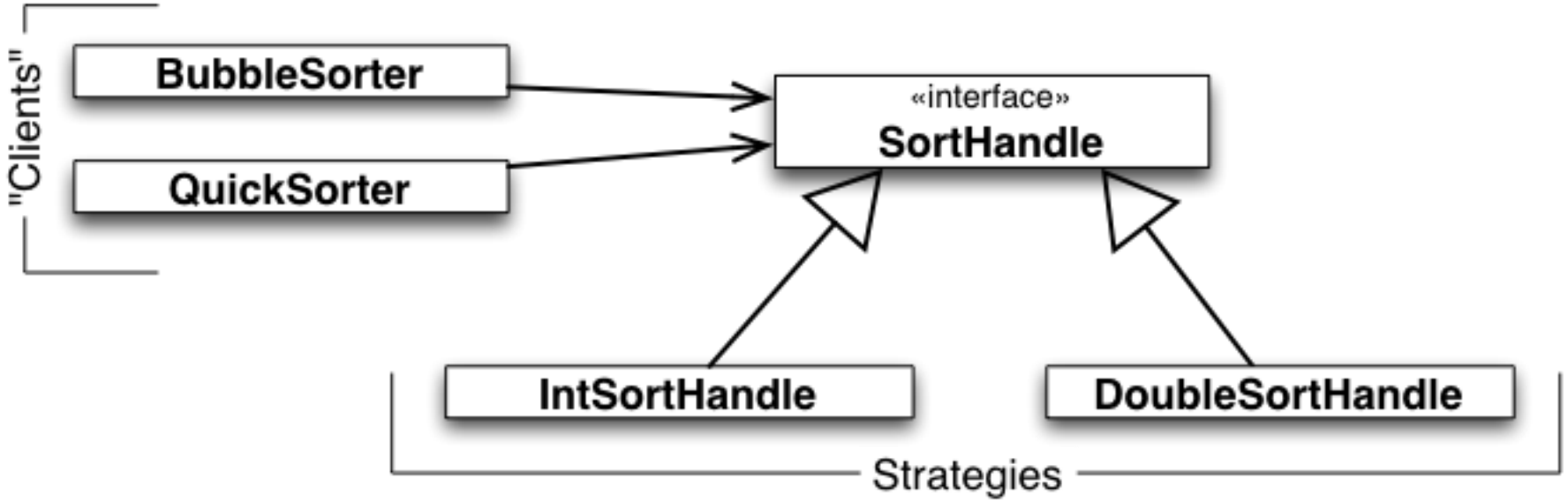
IntBubbleSorter

```
public class IntBubbleSorter extends BubbleSorter{
        private int[] array = null;
        public void sort(int[] theArray) {
                array = theArray;
                length = array.length;
                super.sort();
        }
        protected void swap(int index) {
                int temp = array[index];
                array[index] = array[index+ 1];
                array[index+1] = temp;
        }
        protected boolean outOfOrder(int index) {
                return (array[index] > array[index+ 1]);
        }
}
```

Strategy Pattern



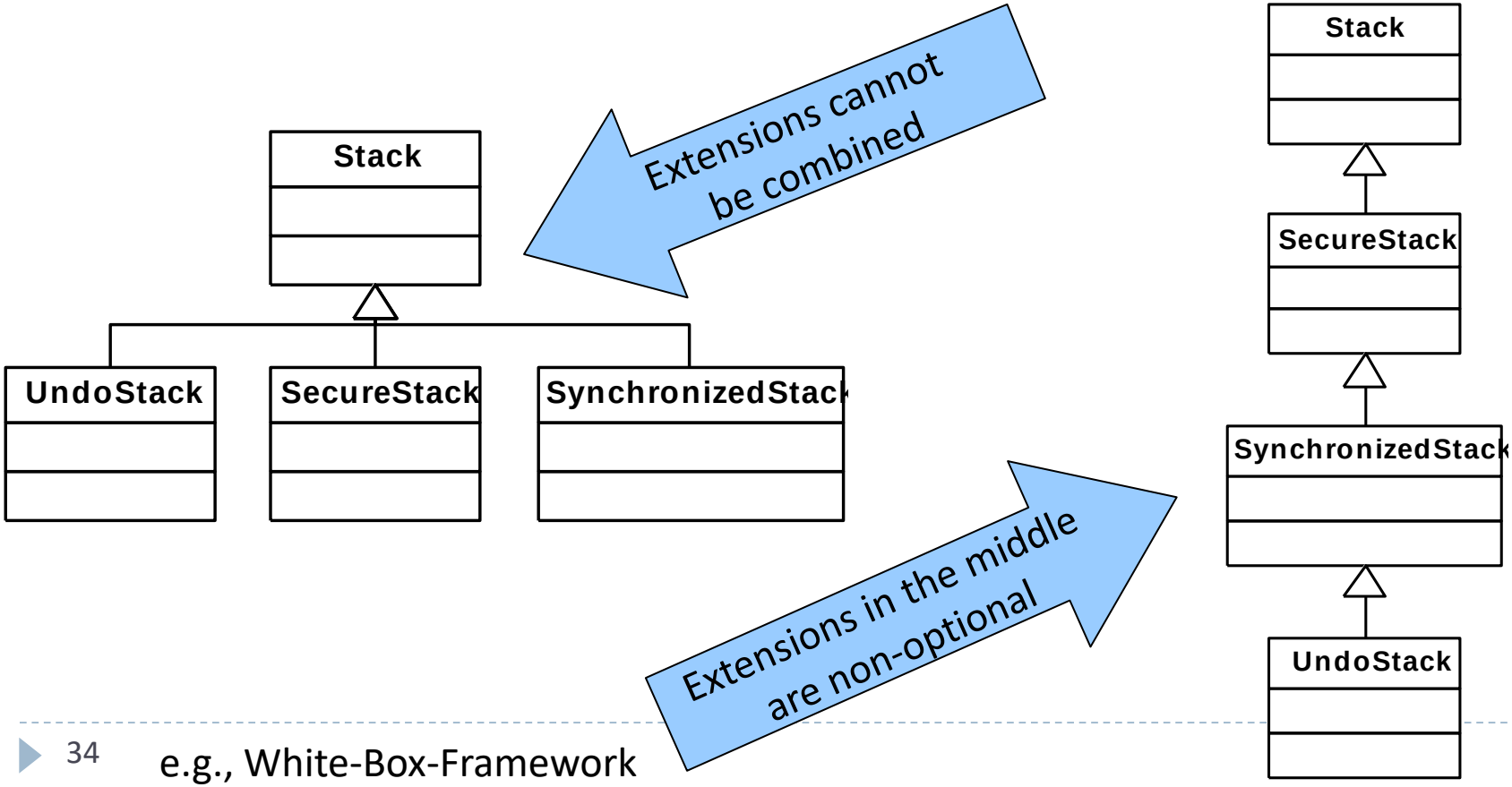
Strategy Pattern: Example



Problem: Unflexible Extension Mechanism

Unflexible Extension Mechanism

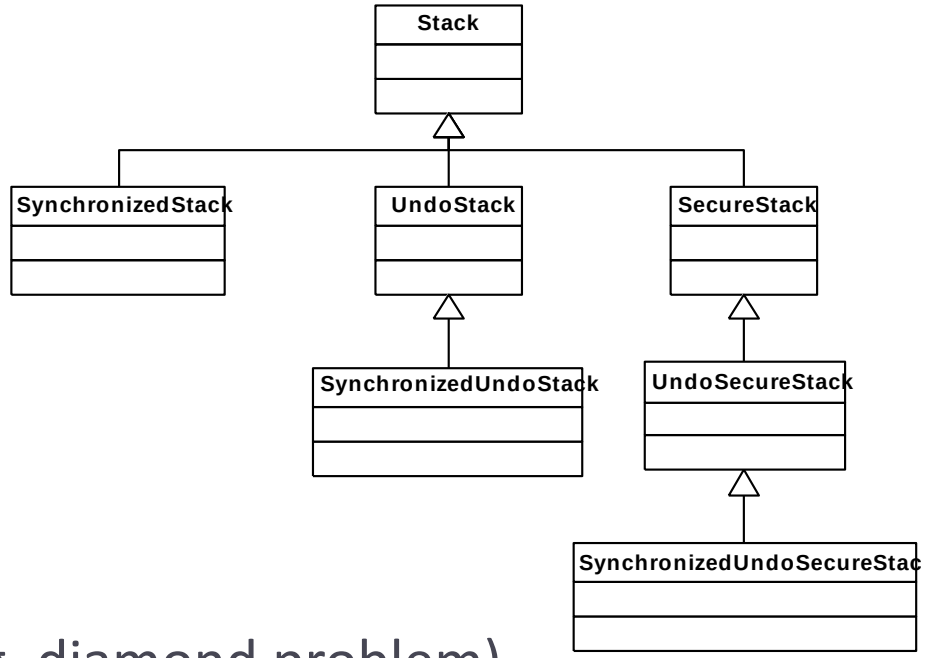
- Subclasses for extensions: modular, but unflexibel
- No “mix & match”



e.g., White-Box-Framework

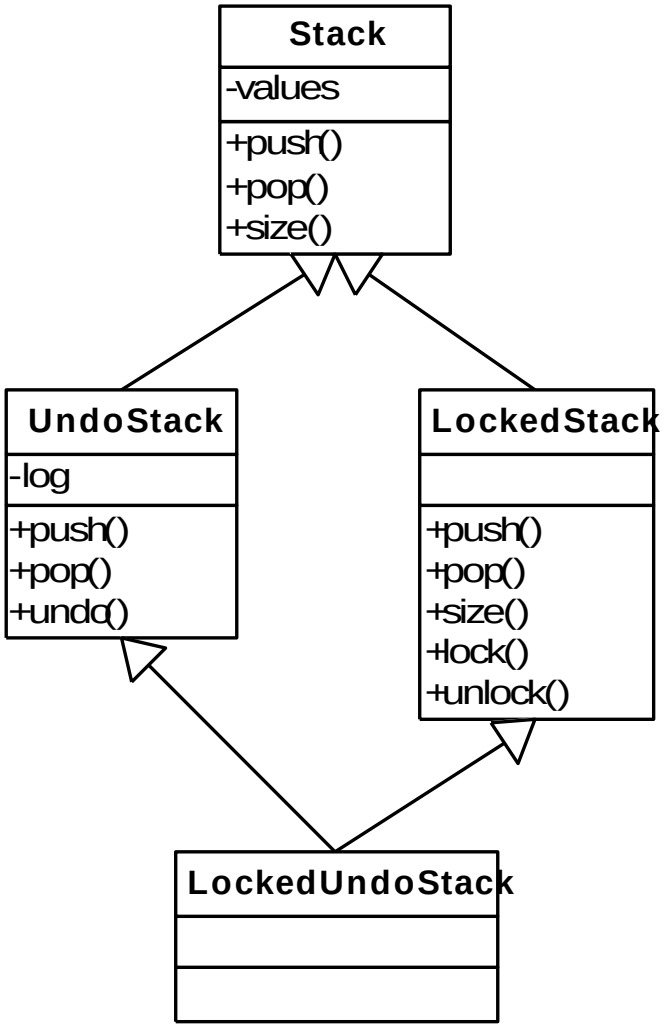
Solution I

- Combined classe hierarchies
  - Combinatorial explosion of variants
  - Massive code replication



- Multi-Inheritance
- Combinatorial explosion
  - Due to diverse problems (e. g. diamond problem) available only in few languages

Diamond Problem



What happends?

```
new LockedUndoStack().pop()
```

"Multiple inheritance is good, but there is no good way to do it."
A. SYNDER

Delegation instead of Inheritance

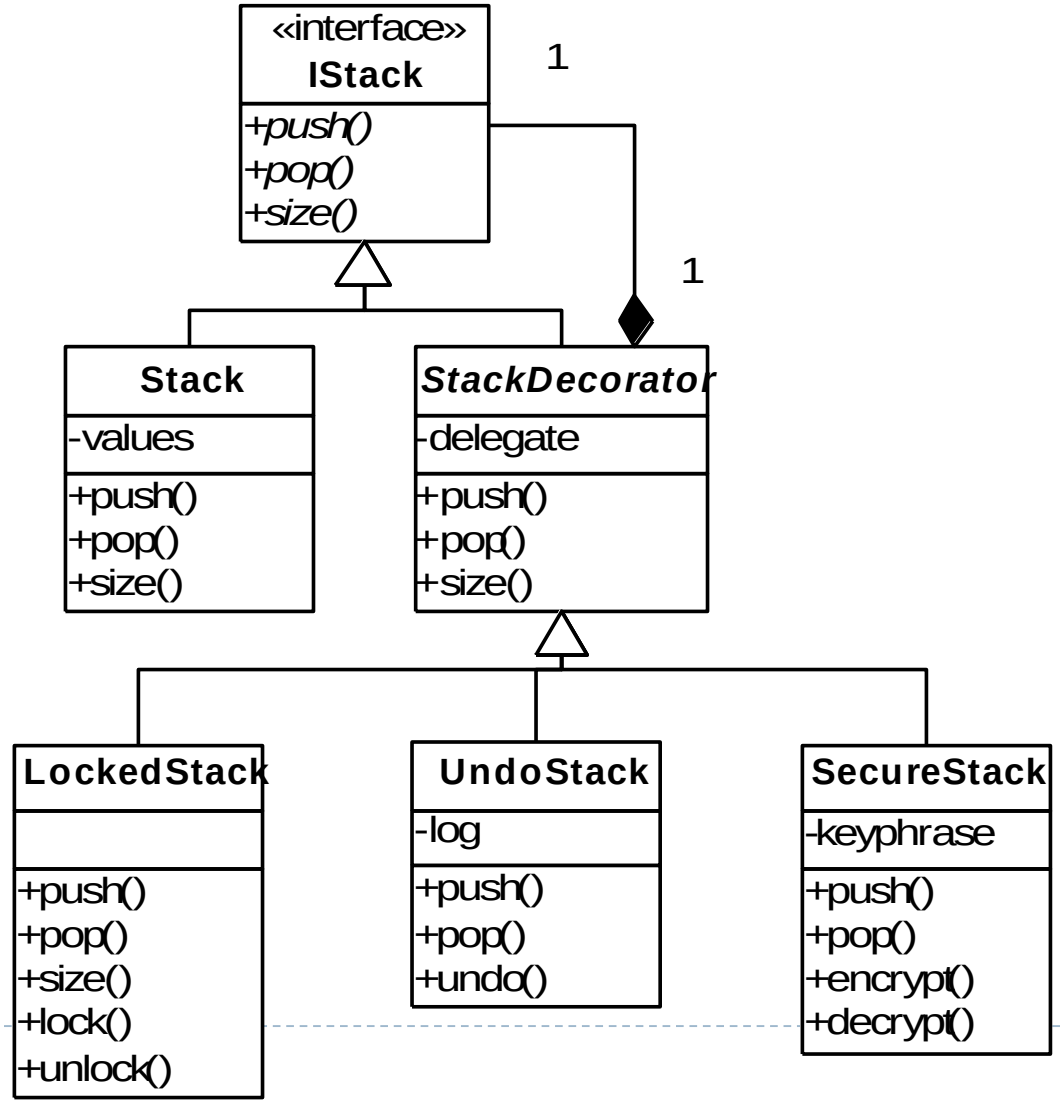
```
class LockedStack implements IStack {
  final IStack _delegate;
  public LockedStack(IStack delegate) {
    this._delegate = delegate;
  }
  private void lock() { /* ... /* }
  private void unlock() { /* ... /* }
  public void push(Object o) {
    lock();
    _delegate.push(o);
    unlock();
  }
  public Object pop() {
    lock();
    Object result = _delegate.pop();
    unlock();
    return result;
  }
  public int size() {
    return _delegate.size();
  }
}
```

```
class UndoStack implements IStack {
  final IStack _delegate;
  public UndoStack(IStack delegate) {
    this._delegate = delegate;
  }
  public void undo() { /* ... /* }
  public void push(Object o) {
    remember();
    _delegate.push(o);
  }
  public Object pop() {
    remember();
    return _delegate.pop();
  }
  public int size() {
    return _delegate.size();
  }
}
```

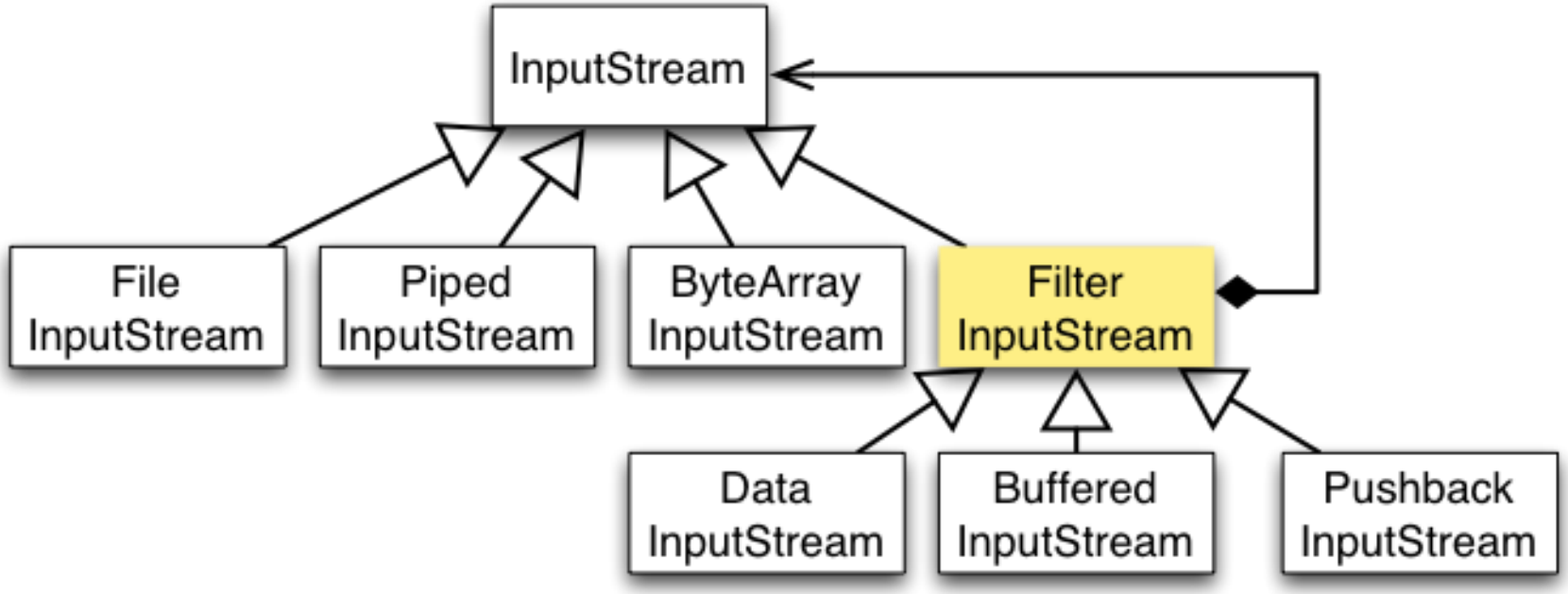
Main:

```
IStack stack = new UndoStack(
    new LockedStack(new Stack()));
```

Decorator Pattern



Example: Decorator in java.io



- java.io contains different functions for input and output:
  - Programs **operate on stream objects** ...
  - **Independent** from the data source / target and type of data

Delegation instead of Inheritance– Discussion

- Dynamic combination is possible
- Extension must be independent from each other
- No way to add new methods (only extending existing ones is possible)
- No late binding (no virtual methods)
- Many indirections in the execution (Performance)
- Multiple instances (objects) form a single object (object schizophrenia)

Flexible Extension Mechanisms?

???

Outlook

- Configuration with generation at compile time
- More flexible extension mechanisms
- Modularising of features

Literature

- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.
  [Standard book for Design Patterns]
- Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1997 – Chapters 3, 4
  [Modularity]

Quiz

- With what design pattern could you implement feature color modularly? (explain)

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

7

- Visualize the module that is
  - (a) very cohesive + strongly coupled
  - (b) very cohesive + weakly coupled
  - (c) low cohesive + strongly coupled
  - (d) low cohesive + weakly coupled